*The International Workshop*
*on Discrete-Event System Design, DESDes'01,*
*June 27÷29, 2001; Przytok near Zielona Gora, Poland*

# ON ALGORITHMS FOR DECYCLISATION OF ORIENTED GRAPHS

Andrei KARATKEVICH

Computer Engineering and Electronics Institute, Technical University of Zielona Góra,
ul. Podgórna 50, 65-246 Zielona Góra, POLAND, *A.Karatkevich@iie.pz.zgora.pl*

***Abstract.*** *The task of decyclization of the oriented graphs is considered. It can be applied to analysis of Petri nets, evaluation of circuits and programs etc. The known methods are reviewed; but they are not effective for big graphs. For the applications exact minimisation of the number of removed arcs usually is not necessary, so a quick heuristic algorithm allowing obtaining an approximate solution would be useful. Such algorithm is described in the paper. The exact methods are also discussed.*

***Key Words.*** *Oriented graphs, circuits, verification, combinatorial tasks*

## 1. BACKGROUND AND THE PROBLEM STATEMENT

Oriented acyclic graphs are widely used in applications (in logical design algorithms, for example), being a convenient way of specifying partial order. If partial order is required, it is necessary to be sure whether the graph is acyclic or it has oriented cycles (circuits). If it has circuits, it may be necessary to transform it into an acyclic graph by removing some arcs; and usually it is better to remove minimal number of arcs enough to destroy all the circuits. So the task [1] is formulated as follows.

For given oriented graph remove minimal number of arcs to obtain acyclic graph.

An alternative formulising: for given oriented graph find maximal acyclic spanning subgraph.

This operation is called *minimal decyclisation* of oriented graph. Finding any acyclic spanning subgraph is called *decyclisation*.

Decyclisation can be applied to analysis of electronic circuits (the minimal set of arcs removing all the circuits of graph is known in electrical engineering as *minimal feedback arc set*). Feedback is a difference between combinational and sequential logical circuits, and the algorithms of logical design and analysis are very different for those two kinds of circuits; of course the algorithms for combinational circuits are much simpler. So selecting of maximal combinational sub-circuit of given circuit can be useful for circuit minimisation, test generation and other CAD and analysis tasks. Decyclisation also can be used for analysis of algorithms represented by graph models, such as PERT (*program evaluation and review technique*), graph-schemes or Petri nets; for disjunction of feedback in the control systems. The author encountered this problem developing algorithms of optimal simulation of Petri

nets [6]. One more area of application is deductive logic, where circuit in graph means a vicious circle.

## 2. MAIN DEFINITIONS

An *oriented graph*, or *orgraph G=(V, E)* is a set *V* of *nodes* together with a set *E* of *arcs* which are ordered pairs of the elements of *V*. The arc *e* is said to be *directed from* its *initial node* init(*e*) *to* its *terminal node* ter(*e*). An orgraph can be specified by its *adjacency matrix* **R** that is a square Boolean matrix n×n, where n=|*V*|; for that matrix $a_{ij}$=1 if and only if there is an arc from node *i* to node *j*.

Number of incoming arcs for node *v* is its *input degree* and is denoted as id(*v*); number of outgoing arcs is its *output degree* and is denoted as od(*v*) [2].

A *circuit* in an orgraph is a path along the arcs of the graph originating and terminating at the same node. An *elementary circuit* is a circuit coming through every node no more than once. Below saying "circuit" we always mean "elementary circuit". A *circuit matrix* **C** is a Boolean matrix m×p, where m – number of circuits, p=|*E*|; for that matrix $a_{ij}$=1 if and only if the arc *j* is in circuit *i* [5].

## 3. THE METHODS OF MINIMAL DECYCLISATION

The known methods of decyclisation of orgraphs are summarised in [1] (see also [8]). Also the effects of removing arcs from orgraphs are discussed in details in [4], but the tasks considered there are slightly different from the task we are interested in. The methods of minimal decyclisation exist, but they cannot be implied to big graphs because they need exponential time. The main approaches are listed below.

### *3.1. Method Using Adjacency Matrix*

In [1] it is shown that an orgraph is acyclic if and only if its nodes can be ordered in such a way that its adjacency matrix becomes strictly triangular (without non-zero elements on the diagonal; such ordering can be obtained by topological sorting of the graph [10]). One of the methods of decyclisation is based on this affirmation. Its main idea follows: transform the adjacency matrix by exchanging columns (and corresponding rows) to make it as close to a strictly triangular matrix as possible (i.e. with minimal number of non-zero elements below the main diagonal or on it). Obtaining the optimal solution here is a difficult combinatorial task; of course some heuristics can be used here to obtain an approximate solution. Advantage of this method is that it doesn't require finding of all the circuits in the graph.

### *3.2. Method Using Boolean Equations*

Another method is described in [1]: let's find all the circuits in the given graph, and let's associate a Boolean variable with each arc. Then for every circuit build disjunction of the correspondent variables (without negation) and construct conjunction of those disjunctions. The obtained Boolean formula is in the conjunctive normal form. Transform it into disjunctive normal form by usual transformations and simplifications of Boolean equations. Every conjunction in the resulting DNF represents a set of arcs, removing of which is enough to destroy all the circuits. The shortest conjunction represents the minimal set of arcs to be removed.

### *3.3. Reducing to the Task of Boolean Matrix Covering*

It is easy to see that the method described above requires in fact finding all the possible sets of arcs enough for decyclisation (minimal in the sense that no element can be removed from those sets). That means that the method is applicable only for very small graphs. But since the set of all circuits in the graph is obtained, a better approach can be used here. As far as there is a set of circuits and it is necessary to select a minimal set of elements belonging to all the circuits, the task is directly reduced to the task of covering of Boolean matrix. So the general algorithm may consist of the next steps:

***Algorithm 1*** *(exact)*

1. Find all the circuits in the orgraph.
2. Construct the circuit matrix **C**.
3. Find the minimal covering of the matrix (minimal subset of columns such that every row has at least one non-zero element in one of those columns).
4. Remove from the graph all the arcs corresponding to the rows of the covering.

Finding the minimal matrix covering is a complex combinatorial task, but this approach is much more effective than the previous one, because there is no necessity to exhaust all the combinations of rows to find the optimal solution. However if finding optimal solution takes too much time, some heuristics may be used; exact and approximate methods of matrix covering are explained in details in [13].

The algorithm finding all the circuits in an orgraph is described in [11] (firstly in [3]). Both tasks (finding all circuits and minimal matrix covering) are NP-complete (the number of circuits in a graph exponentially depends on its size), so the only practical way of solving the task for a big graph is using heuristics.

## 4. THE HEURISTIC ALGORITHM

### *4.1. Analogy between the Tasks of Decyclisation and Finding of Spanning Trees*

The task of finding of spanning tree [11] is usually considered for non-oriented graphs. The analogy between this task and decyclisation exists because if non-oriented cycles are considered, an ayclic graph turns to be a tree. In this sense, finding maximal acyclic spanning subgraph for an orgraph can be considered as a generalisation of the task of finding spanning tree for a non-oriented graph. But all the spanning trees for given graph have the same number of arcs; so the question of optimisation arises if the arcs are weighted and their total weight should be minimised or maximised. For the orgraphs that is not the case; different spanning subgraphs with different number of arcs may be maximal in the sense that no arc can be added to them without creating a circuit (Fig. 1; dashed arrows denote the removed arcs). But, as it is shown below, the approach used for finding minimal (or maximal) spanning tree can be used with some changes for finding maximal acyclic spanning subgraph of an orgraph.

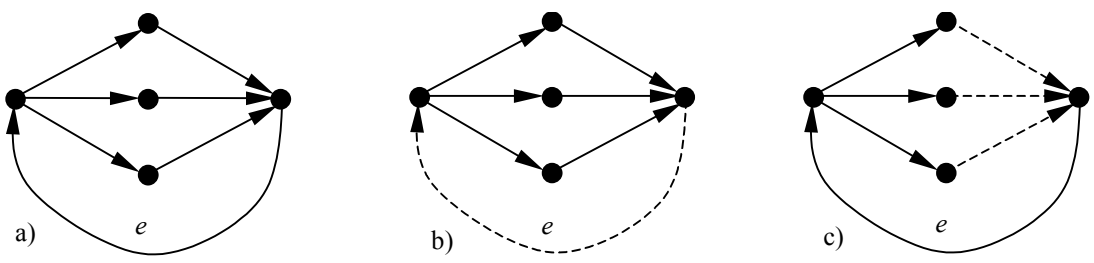


*Figure 1. An orgraph (a) and its different acyclic subgraphs (b,c)*

*4.2. The Main Idea and Heuristic*

An effective heuristic algorithm for decyclisation cannot require finding all the circuits in the given graph (a NP-complete task). So it is intuitively clear that a heuristic algorithm can build gradually an acyclic subgraph of the given graph. The same approach is used in [7] (Kruskal algorithm); but for a weighted non-oriented graph and the task of minimal spanning tree finding it allows obtaining the optimal solution, and for our task the solution will be approximate. The idea is, to start from any node of the given graph and to build an acyclic graph by adding the arcs such that the subgraph remains acyclic. But to direct this process and to avoid some non-optimal variants, the weights should be associated with the arcs of the given graph. We need a simple formula of calculating weights. And the evident way to do it is to use input and output degrees of initial and terminal nodes for each arc.

How to use these data? Let's consider some examples. It is easy to see that we should avoid adding to the spanning subgraph the arcs like the arc *e* at Fig. 1. That is an arc belonging to many circuits, so its presence in the subgraph would mean that several other arcs cannot simultaneously be there, and the obtained solution will be far from optimal. We don't know however the number of circuits each arc belongs to. (If it would be known the approach of removing the arcs belonging to the maximal number of circuits would be equivalent to one of the heuristic methods of matrix covering; but building the circuit matrix would require finding all the circuits). But what can be said about it by analysing input and output degrees?

If the terminal node has many outgoing arcs, and the initial node has many incoming arcs, that means that it belongs to many paths (and maybe circuits), and probably that it is the only arc common to all those paths (or one of the few ones). That means that it is undesirable to add this arc to the spanning subgraph. On the other hand, if the terminal node has many incoming arcs, and the initial node has many outgoing arcs, we may suppose that there are many circuits and if they have common arcs then somewhere else; so the arc under consideration can be added to the subgraph.

This intuitive reasoning can be expressed by the next formula:

$$w(e) = id(init(e)) - od(init(e)) + od(ter(e)) - id(ter(e)) \qquad (1)$$

where *e* is an arc and *w(e)* is its weight.

Then it is enough to create a spanning subgraph starting from the arc with minimal weight and adding also the arcs with minimal weights such that the subgraph remains acyclic. When no arc can be added to the subgraph, removing the rest of arcs from the given orgraph solves the task of decyclisation.

Here it is necessary to check at each step whether the subgraph is acyclic. It is an easy task for both oriented and non-oriented graphs; a graph has a cycle if and only if its depth-first-search tree contains a feedback arc. It can be checked by transitive closure of the relation specified by the given graph [12].

*4.3. Formal Description of the Algorithm*

***Algorithm 2*** *(heuristic)*

Let $G=(V, E)$ be the given graph. $G'=(V', E')$ such that $V'=V$, $E'=\varnothing$. $S=\varnothing$.

1. While $\exists\, v\in V$: $(\mathrm{id}(v)=0 \vee \mathrm{od}(v)=0)$

   1.1. $E \leftarrow E\backslash\{e\in E: v\in e\}$.*

---

* Such operation is useful also for exact algorithm; the removed arcs certainly don't belong to any circuit.

2. Calculate for every arc $e \in E$ its weight according to (1).
3. While ($E \setminus S \neq \varnothing$)

   3.1. $e \leftarrow \min(w(e): e \in (E \setminus S))$;

   3.2. $S \leftarrow S \cup \{e\}$;

   3.3. $E' \leftarrow E' \cup \{e\}$;

   3.4. If $G'$ has a cycle then $E' \leftarrow E' \setminus \{e\}$.
4. ($E \setminus E'$) is a set of arcs, removing of which is enough for destroying all circuits in $G$. The end.

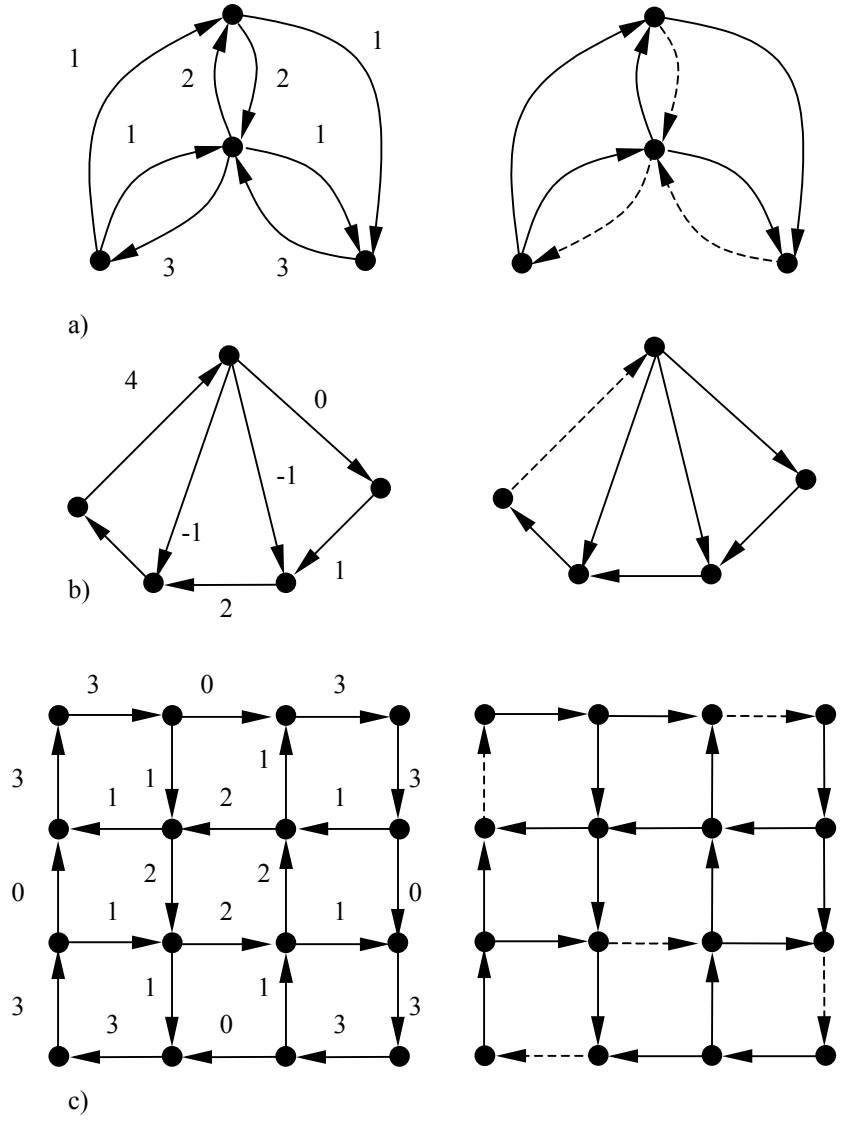


*Figure 2. Examples of decyclisation*

## 5. EXAMPLES

Let's consider some examples (Fig. 2). The graphs are taken from [4]. In all those cases the approximate algorithm finds the same solution as the exact one, i.e. the optimal solution. Weights of the arcs are shown near the arcs at Fig. 2. Note that we calculate weight for an arc not counting the arc itself for defining input and output degrees; so all the weights at Fig. 2

are 2 less than according to (1). That seems to be more convenient and doesn't affect to the results.

We can see that arcs with maximal weights can be used as the important indication for decyclisation. Consider Fig. 2a. Gradual building of acyclic spanning subgraph goes on as follows: at first the arcs with weight 1 are added to *G'*; it remains acyclic. Then one arc with weight 2 is added there; *G'* is still acyclic but no more arc can be added to it. So the decision is found, and it is optimal. Similar situation is with graphs shown at Fig. 2b and 2c; they are not explained in details because of lack of space.

## 6. CONCLUSIONS

The suggested approach allows quick obtaining of good approximate solutions of the task of decyclisation of oriented graphs. It can be applied to various problems such as Petri net analysis, minimisation and test generation for sequential circuits (by decyclisation these tasks can be reduced to the similar tasks for combinational circuits) and others, mentioned in *Background and the Problem Statement*. Statistical analysis of the method effectiveness is a topic of future work.

## REFERENCES


[1] N.Deo, *Graph theory with applications to engineering and computer science*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, USA, 1974.

[2] R.Diestel, *Graph Theory.* Electronic Edition, Springer-Verlag, New York, 2000.

[3] D.B.Johnson, *Finding All the Elementary Circuits of a Directed Graph.* SIAM J. Comput, 4, 1975, pp. 90-99.

[4] F.Harary, R.Z.Norman, D.Cartwright, *Structural Models: An Introduction to the Theory of Directed Graphs.* New York: John Wiley, 1965.

[5] E.J.Henlog, R.A.Williams, *Graph Theory in Modern Engineering: Computer Aided Design, Control, Optimization, Reliability Analysis.* Academic press, New York, 1973.

[6] A.Karatkevich, "Optimal Simulation of α-Nets". *Proceedings of the Polish-German Symposium on SRE'2000*, Zielona Góra, 2000.- P. 217-222.

[7] J.B.Kruskal, *On the shortest spanning subtree of a graph and the travelling salesman problem.* Proc. Amer. Math. Soc., 7, 1956, pp. 48-50.

[8] A.Lempel, "Minimum Feedback Arc and Vertex Sets of a Directed Graph". *IEEE Trans. Circuit Theory*, 1966 Dec., Vol. CT-13 No 4, pp. 399-403.

[9] E.M.Reingold, J.Nievergelt, N.Deo, *Combinatorial Algorithms: Theory and Practice.* Prentice-Hall, Inc., New Jersey, 1977

[10] R.Sedgewick, *Algorithms.* Addison Wesley, Inc., USA, 1984.

[11] M.M.Syslo, N.Deo, J.S.Kowalik, *Discrete Optimization Algorithms with Pascal Programs*. Prentice-Hall, Inc., 1983.

[12] S.Warshall, *A Theorem on Boolean Matrices.* J. ACM, 9, 1962, pp. 11-12.

[13] A.D.Zakrevskij, *Logic synthesis of cascade circuits*. Moscow, Nauka, 1981 (in Russian).